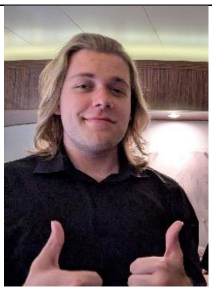
SP-104 Active Learning System Images Development Document

CS 4850 – Section 02 – Fall 2025

Matthew Hall, Josh Smith, Elijah Merrill, Noah Lane

Josh Smith Team Leader Developer	Elijah Merrill Developer	 Noah Lane Documentation	Matthew Hall Documentation

Team Members:

Name	Role	Cell / Alt Email
Josh Smith (Team Lead)	Developer	706-414-2827 joshuasmith0515@gmail.com
Elijah Merrill	Developer	478-283-0811 elijahmerrill04@gmail.com
Matthew Hall	Documentation	678-873-8542 thematthewhall7@gmail.com
Noah Lane	Documentation	706-591-2312 noahlane142@gmail.com
Sharon Perry	Advisor	770-329-3895 sperry46@kennesaw.edu

1.0 Technical Details

The first major process in development is the implementation of the dataset. The specifics of the code files will be discussed here while the overall database connection will be discussed in the following segment.

1.1 Indexer

The first source code file is indexer.py. This file is responsible for constructing a structured table that maps each image to its associated metadata. Notable libraries imported are pandas and numpy. These help with numerical calculations and reading tabular data.

The build_index function reads the NIH Chest X-ray metadata file (known as Data_Entry_2017.csv and organizes the information into a unified dataset. This includes linking each image file to certain aspects, like a patient ID and diagnostic labels. The function verifies that all image paths exist, cleans label strings, and structures the data into a DataFrame, which will be used in later phases for training.

Additionally, the filter_views function keeps only the frontal chest X-rays from the dataset. These frontal images are known as Posterior Anterior (PA) and Anterior Posterior (AP) and are fancy terminology for frontal X-ray images when it comes to varying factors like a patient's position and the X-ray beam direction. This will help in later phases to maintain consistency in sampling and training.

1.2 Splitter

The splitter.py code file is used to split the dataset by patient IDs, so that no patient data appears in both training and testing sets. Numpy, pandas, and JSON are imported.

The main function, split_by_patient(), splits these sets into train, validation, and test sets based on the unique patient ID instead of the individual images. The parameters are the dataframe, the ratios for splits, and a seed for random number generation. The core of the function gets all unique IDs and shuffles them randomly. It then calculates how many patients go into each split and divides them into sets. The updated dataframe with a new “split” column is returned.

Implementing the splitter file uses a 70/15/15 ratio for a partitioning model. 70% will go to the training, 15% will go to validation, and 15% will go to the testing section for the project.

Another function called save_splits_json is used to make a quick summary of the results in a JSON file. This function builds a dictionary of the number of rows and unique patients in each split.

1.3 Preprocessor

The preporcess.py code file is used to create a sequence of image preprocessing and augmentation operations for the active learning model. It uses torchvision.transforms to help convey the data to a PyTorch-friendly format.

The core function, make_transforms, constructs a customizable transformation pipeline that applies several processing steps. It begins by initializing an empty list of transformations, which will then be populated when the model is training. For training, random augmentations such as horizontal flipping and rotations can be applied to artificially expand dataset diversity and improve model generalization. Next, all images are resized to a fixed input dimension of 224x224 pixels and converted into PyTorch tensors for numerical representation across batches.

If ImageNet normalization is enabled, the function applies normalization using pre-calculated mean and standard deviation values for chest X-rays. This will ensure compatibility with pretrained neural networks which would expect an input normalized to ImageNet’s color distribution.

Finally, transform.Compose, is used to merge all these individual transformations into a single callable object, allowing for the dataset loader to preprocess each image during model training.

1.4 Dataset

The dataset.py file is responsible for loading and preparing image-label pairs for the active learning model. This file acts as the bridge between the preprocessed data and the training process by integrating data indexing and transformations.

The main class, CXRDataset, inherits from torch.utils.data.Dataset, allowing it to integrate with PyTorch's data pipeline. When initialized, the class receives the indexed dataset from the indexer.py, a specialized data split (either train, validation, or test), a list of class names, and a set of preprocessing transforms to apply to each image.

The __getitem__ method handles how individual samples are accessed. For each index, it loads the corresponding X-ray image using the Pillow (Pil) library, converts it to an RGB format, and applies the transformation pipeline defined in the preprocess file. The transformed image is then converted to a PyTorch tensor, and its label information is extracted and encoded as a numeric tensor representing the presence or absence of each medical condition.

The __len__ function returns the total number of samples available in the current split, which the DataLoader uses to determine iteration length during training.

1.5 Configuration File

The config.yaml file serves as the central configuration hub for the active learning model. It defines all key parameters needed for numerous features, like dataset loading, preprocessing, and data partitioning. This allows for the entire pipeline to be easily reconfigured without the need to change source code.

The configuration is divided into four sections, each of which controls a different stage of the data pipeline.

- Paths: This section defines the file system location used throughout this project. The data_root specifies the directory containing the original dataset of the NIH chest X-ray which contains the images and metadata CSV file. These next two parameters define where the system will save processed outputs. Index_out stores the generated image index table as a parquet file, linking image paths and metadata. Split_out stores the dataset split summary as a .json file, recording how images and patients belong to each split.
- Data: This section lists class names, which are mainly the disease categories the model will detect. The keep_views parameter specifies which X-ray orientations to include, mainly focusing on PA (posteroanterior) and AP (anteroposterior) views as they are frontal chest views.

- Model: Here, the input_size parameter defines the target image resolution that all images are resized during pre-processing. The use_imagenet_norm parameter enables ImageNet-based normalization, scaling pixel intensities according to pre-trained model statistics.

- Splits: This section specifies how the dataset should be divided for training, validation, and testing (70/15/15 split). The seed parameter controls random number generation, guaranteeing that dataset splits remain reproducible, so the same patients are consistently assigned to the same subset across multiple runs.

Together, the config.yaml file enables the system to operate in a fully parameterized manner, separating configuration from implementation.

1.6 Smoke File

The smoke.py file serves as a lightweight integration test for the data processing pipeline. Its primary purpose is to verify that each component, from data indexing and splitting to preprocessing and dataset loading, functions correctly and interacts smoothly before model training begins. This file validates that each file listed so far within this document operates smoothly when imported and that developers can detect various issues before training implementation begins.

1.7 Frameworks/Libraries Used

Pandas - An open-source Python library primarily used for data manipulation and analysis. Plays a key role in reading and organizing metadata from the NIH dataset's Data_Entry_2017.csv file. Pandas DataFrames are used throughout the pipeline to store various parameters in a structured format for processing.

NumPy – A core library used for array operations that support data indexing and transformations when splitting datasets. Integrated through the indexer.py and splitter.py modules.

Pillow – An imaging library within Python used for image loading and conversion. It allows reading of X-ray image files in various formats and converting them into RGB format. This library can be found within the dataset.py module.

TorchVision – A PyTorch companion library focused on image transformations and augmentations, and preparing the images for CNN input in preprocess.py.

TQDM – A progress bar utility for Python loops. It will be useful for monitoring progress during model training and batch processing in later project phases.

PyTorch – Core module framework for this project. A deep learning framework used to build and train neural networks. While not yet implemented within this first phase, the system is designed to integrate PyTorch's Dataset and DataLoader classes to handle batch training and real-time data feeding into a CNN architecture (such as ResNet or DenseNet) in future phases.

OS/Pathlib – Built-in python libraries for file and path management. They ensure the program can dynamically locate image files during indexing and loading.

2.0 Database Connection – Not Applicable

This project does not implement a database connection but instead employs a file-based data storage approach with a dataset that is downloaded on the user's system. This project does not connect to a traditional database. Instead, it utilizes a publicly available chest X-ray dataset stored and managed locally within the system. The dataset serves as the primary data source for model training, validation, and testing. All image files and corresponding labels are loaded directly from local storage or structured folders rather than through a database connection.

3.0 Set-up Process

In the GitHub repository, there are a total of six files that are essential to the overall functionality of the system. The files that need to be downloaded to the user's system can be found in the "source code" folder within the repository. It is highly recommended to keep track of where these source code files are stored for easier configuration and troubleshooting.

The next important component is the dataset itself, which can be found on Kaggle at the following link: https://www.kaggle.com/datasets/nih-chest-xrays/data. First-time users will likely need to create a Kaggle account to access and download the dataset. Once downloaded, the dataset will be contained in a folder named "archive." It is recommended to store this folder in the same directory as your source code files. If that is not possible, make sure to record the exact file path for later reference.

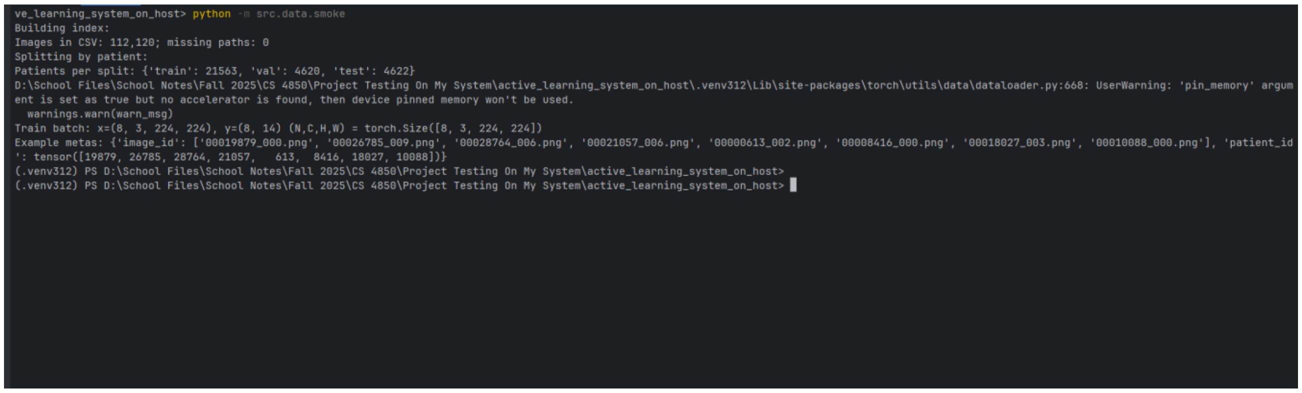
A Python version of 3.10 or later is required to compile and run this system. The latest Python version can be downloaded from https://www.python.org/downloads/. Several dependencies and packages must also be installed for the system to function properly. The primary one is PyTorch, which can be installed by running the command "`pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu124`" in the PowerShell terminal. Further instructions or troubleshooting information can be found on the official PyTorch website at https://pytorch.org/.

Additional required packages include pandas, numpy, and pyarrow, which can be installed using the command “`pip install pandas pyarrow numpy`.” Possible packages that will be implemented include tqdm and matplotlib, which can be installed using “`pip install tqdm matplotlib`.”

After all dependencies are installed, ensure that the dataset folder (archive) and source files are properly configured and that all file paths match your local directory structure. If the dataset or output folders are stored in non-default locations, update the script variables or configuration entries that reference them, such as those specifying archive/Data_Entry_2017.csv or the output directories for generated files.

To verify that everything is set up correctly, execute the script “smoke.py” by running “`python smoke. py`” in the terminal. This script reads the dataset metadata and generates an indexed output file named index_table.csv, and optionally index_table.parquet, inside the “outputs” directory. Successful creation of these files confirms that the dataset, file paths, and environment are properly configured, and that the system is ready for further training or evaluation.

Here is what the output should look like to help validate successful implementation with all the files.



```
ve_learning_system_on_host> python -m src.data.smoke
Building index:
Images in CSV: 112,120; missing paths: 0
Splitting by patient:
Patients per split: {'train': 21563, 'val': 4620, 'test': 4622}
D:\School Files\School Notes\Fall 2025\CS 4850\Project Testing On My System\active_learning_system_on_host\.venv312\Lib\site-packages\torch\utils\data\dataloader.py:668: UserWarning: 'pin_memory' argum
ent is set as true but no accelerator is found, then device pinned memory won't be used.
  warnings.warn(warn_msg)
Train batch: x=(8, 3, 224, 224), y=(8, 14) (N,C,H,W) = torch.Size([8, 3, 224, 224])
Example metas: {'image_id': ['00019879_000.png', '00026785_009.png', '00028764_006.png', '00021057_006.png', '00000613_002.png', '00008416_000.png', '00018027_003.png', '00010088_000.png'], 'patient_id
': tensor([19879, 26785, 28764, 21057,   613,  8416, 18027, 10088])}
(.venv312) PS D:\School Files\School Notes\Fall 2025\CS 4850\Project Testing On My System\active_learning_system_on_host>
(.venv312) PS D:\School Files\School Notes\Fall 2025\CS 4850\Project Testing On My System\active_learning_system_on_host> 
```